Atlas Fusion - Modern Framework for Autonomous Agent Sensor Data Fusion

LIGOCKI, A.; JELÍNEK, A.; ŽALUD, L.

14th International Conference ELEKTRO, ELEKTRO 2022 – Proceedings

ISBN: 978-1-66-546726-1

DOI: https://doi.org/10.1109/ELECTRO53996.2022.9803587

Accepted manuscript

Atlas Fusion - Modern Framework for Autonomous Agent Sensor Data Fusion

1st Adam Ligocki
Cybernetics and Robotics, CEITEC
Brno University of Technology
Brno, Czechia
0000-0002-6813-4318

2nd Aleš Jelínek
Cybernetics and Robotics, CEITEC
Brno University of Technology
Brno, Czechia
0000-0001-7519-2092

3rd Luděk Žalud
Cybernetics and Robotics, CEITEC
Brno University of Technology
Brno, Czechia
0000-0001-5996-8137

***Abstract*—In this paper, we present our software sensor fusion framework for self-driving cars and other autonomous robots. We have designed our framework as a universal and scalable platform for building up a robust 3D model of the agent's surrounding environment by fusing a wide range of various sensors into the data model that we can use as a basement for the decision making and planning algorithms. Our software currently covers the data fusion of the RGB and thermal cameras, 3D LiDARs, 3D IMU, and a GNSS positioning. The framework covers a complete pipeline from data loading, filtering, preprocessing, environment model construction, visualization, and data storage. The architecture allows the community to modify the existing setup or to extend our solution with new ideas. The entire software is fully compatible with ROS (Robotic Operation System), which allows the framework to cooperate with other ROS-based software. The source codes are fully available as an open-source under the MIT license. See https://github.com/Robotics-BUT/Atlas-Fusion.**

***Index Terms*—Open Source, Autonomous Agent, Self Driving Car, Sensor Fusion, Mapping, ROS**

I. Introduction

As the world is diving deeper into the problem of self-driving cars and other autonomous robots, there is a large number of sophisticated systems for analyzing data and controlling the specific problems of autonomous behavior. However, these systems, like [1] or [2] are very complex and require dozens of hours to understand the architecture and to be able to start to develop a new solution on top of the existing one.

As members of the academic community, we are experimenting with many new approaches. Our primary motivation is to search for new ways and improve the current state-of-the-art techniques. For this purpose, we designed a system aiming at surrounding environment sensing and map building in mobile robotics.

The work has been performed in the project NewControl: Integrated, Fail-Operational, Cognitive Perception, Planning and Control Systems for Highly Automated Vehicles, under grant agreement No 826653/8A19006. The work was co-funded by grants of Ministry of Education, Youth and Sports of the Czech Republic and Electronic Component Systems for European Leadership Joint Undertaking (ECSEL JU) The work was supported by the infrastructure of RICAIP that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306 and from Ministry of Education, Youth and Sports under OP RDE grant agreement No CZ.02.1.01/0.0/0.0/17_043/0010085.

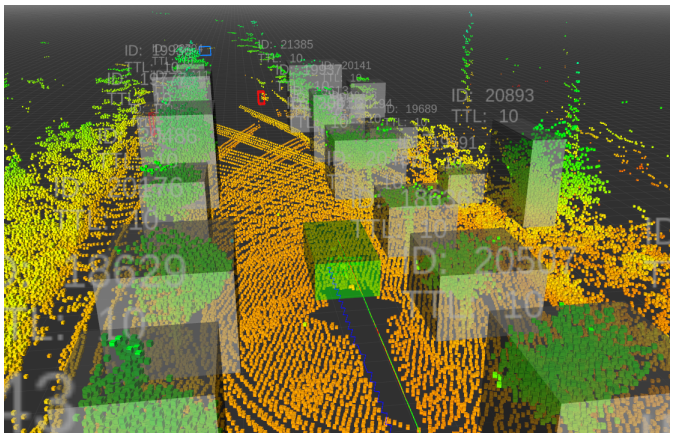

Fig. 1. The example of the RViz visualization of the runtime model of the surrounding environment. Grey boxes are the LiDAR-based detections, and color frustums are the RGB images' neural network detections. The green object at the center represents the agent and the lines behind the agent are the trajectories estimated by different filtering algorithms.

As a result, our team created this C++ framework focusing on data fusion from the various sensor types into a robust representation of the robot's surroundings model.

II. General Architecture Description

We have designed the software with the idea of a very minimalistic pipeline and simple modification to develop and deploy new ideas and algorithms quickly.

A. Input Data

As an input data format, we have chosen the same representation used previously in our work on Brno Urban Dataset [3], which is inspired by [4].

The data are stored as an h265 video in case of RGB and thermal camera data, .ply files for LiDAR scans, and CSV data files for GNSS, IMU, and camera and LiDAR timestamps.

B. Core Pipeline

At the startup, the program reads the basic configuration from the config file. The configuration provides a path to the offline record, and the data loading module loads up all the

necessary information for offline data interpreting. After that, the main pipeline begins.

The loading module loads all timestamped data into the memory and later provides the data in the correct order, one by one. The pipeline redirects data into the dedicated processing section based on the data type and from which sensor the data comes. The output data, like detected obstacles, static obstacles, or moving entities, are stored in the local map data model.

The entire pipeline has a linear architecture, so the data processing algorithms are sorted one by one. This waterfall-like design allows anybody to add or remove a new data processing algorithm without affecting the current ones.

C. Outputs

The framework's main output is the map of the surroundings, stored in the Local Map block, with the precise detection of the possible static and dynamic obstacles. The following decision-making algorithms can use this map to adjust the agent's behavior based on the mapping process's data.

Secondary, there are several other outputs described in detail in section IV. There are the 3D models of all the places that the agent visited during the mapping session, projections of neural network's detection from RGB cameras to thermal images (an annotated IR dataset for object detection is created this way), and the depth maps for camera images generated from the aggregated point clouds.

III. Modules

A. Data Loaders

As our framework is currently not working with online data, there is an interface that loads stored records and provides the loaded data ordered by their timestamps to the main pipeline.

There is a data loader for every physical sensor that reads only one data series. These data loaders are wrapped by a central data loader that creates an interface between stored data and the main pipeline. All the data loaders have ordered the timeline of their data series. When the main pipeline is ready to accept the next data packet, the central data loader asks all the loaders for their smallest timestamp. The data loader with the lowest timestamp will provide the data packet to the processing pipeline.

B. Data Models

The first part of the data models is the raw input data representation. Every sensor has one or more classes that cover the range of the input data. For example, a camera. There are two classes CameraFrameDataModel for RGB image representation and the CameraIRFrameDataModel for the thermal camera image data entity. Every instance of those classes defines the camera sensor identifier, precise timestamp, image frame, and optionally pre-generated YOLO neural network object detections. This data packet keeps all the important information, and the data loader passes the instance of this class when the main processing pipeline requests the latest image data.

The second part of the data models is the internal data representation models used for communication between the modules in the primary data processing pipeline. For example, the LidarDetection structure for objects detected in the LiDAR domain, LocalPosition as a relative metric position w.r.t. the origin of the mapping session, FrustumDetection for the camera-based detected objects and many others.

C. Algorithms

The "Algorithms" module is the core one. It contains all the data processing code. Here the implemented classes cover the agent's position filtration based on Kalman fusion of the GNSS and IMU inputs, functionality for projecting objects from the 3D environment into the camera frames and back, generating a depth map from the LiDAR data, or the redundant data filtration. The "Algorithms" module is the main section where the implementation of the pipelines is described in Section IV.

D. Local Map

The "Local Map" module primarily represents the software that holds the internal map of the surrounding environment. There are two main classes. The first one is LocalMap. This class is a simple container that allows us to store and read out data models of the map representation entities, like aggregated LiDAR model of the near surrounding, detected obstacles, YOLO detections, and higher representations of the more complex fused data. The second class is ObjectsAggregator. This class fuses low complexity detections, such as LiDAR and camera-based detected objects, into the higher complexity representation, fusing geometrical shape information, object type, kinematic model, motion history, etc.

E. Visualizers

This module handles the interface between the main pipeline and local map, and the rendering engine. The main class, called VisualizationHandler provides a wrapper over the entire rendering logic. For every specific data type (IMU data - ImuVisualizer, camera frames - CameraVisualizer, point clouds - LidarVisualizer, etc.) there is dedicated class that manages the interface between the central point and the visualization engine (RViz in our case).

F. Data Writers

The Data Writer section covers the classes responsible for writing Local Map data to the local hard drive storage. Currently, there are the implementations for saving the aggregated LiDAR point cloud projected to the camera plain (see IV-E) and the class for storing RGB YOLO detections projected into the thermal camera (see IV-D).

IV. Data Processing Pipelines

The framework implements several principles of data processing and map building. In this section, we are describing the basics of the most important ones.

A. Precise Positioning

Without precise positioning, it would be impossible to build a reliable map model and aggregate information in time.

For our purpose, we used the differential RTK GNSS that samples a global position with the precision of one σ below 2cm and provides an azimuth of the measurement setup. To improve the dynamic positioning, we also use the linear acceleration and angular velocity from the IMU sensor. An example of the fusion of these sensors could be [5].

The pipeline has the following input data: the global position and heading from the GNSS receiver, linear acceleration, angular velocity, and filtered absolute orientation from the IMU sensor. The IMU automatically compensates for the roll and pitch drift by the gravity's direction, and the yaw drift compensates by the magnetic field measurement.

In the beginning, the first GNSS position sets up an anchor that defines the mapping session's origin. This first global position is the origin (the anchor) of the local coordinate system. The core of the position estimation process is the set of 1D Kalman filters [6], [7], that model position and speed in all three axes of the given environment. Every new incoming GNSS position is converted to the local coordinate system w.r.t. the anchor. This local position is used as a correction for the Kalman filters [8] in all three axes.

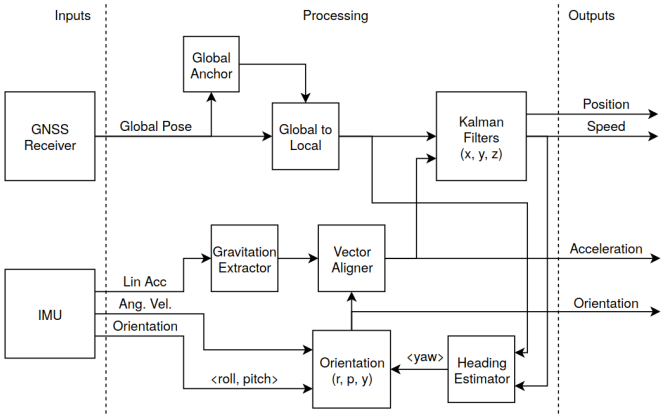

Fig. 2. Scheme of the position estimation pipeline.

As the system models the IMU orientation separately on the IMU's internal model, every new angular velocity data system updates its internal model to have a fast response. However, there is always a long-term drift for this long-term noisy data integration. The system fuses its internal model with the IMU's one using the low pass filter to remove the roll and pitch drifts. To compensate for the yaw drift, it combines the heading measured by the GNSS receiver and its differential antennas with the heading estimated by the agent's speed, which the motion model estimates.

B. LiDAR data aggregation

As we are using the rotating 3D LiDARs, the scanners perform measurements in different directions at different times during the scanner motion, and the robot is constantly changing its position. All these effects cause the outcome measurement to be significantly distorted [9], [10].

Thus, we can not merge all the scans into one because the result would be inaccurate and blurred.

The input LiDAR data could come from several LiDAR scanners. The entire process assumes that each scan stores the data in the same order as it was measured. However, the input data are at the beginning filtered by the data model's callback and downsampled by the PointCloudProcessor call instance to reduce the computational complexity of the later point cloud transformation. At the same time, the positioning system provides the agent's position when the current and the previous scans were taken.

All these three information, the scan, and both positions are passed to the PointCloudExtrapolator instance. There the point cloud is split linearly into the N batches of the same size. Because the scan data are sorted, each batch covers a small angular section of the entire scan, corresponding to the small-time period when the batch data has been taken.

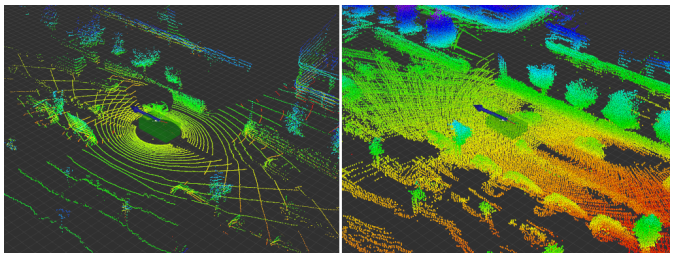

Fig. 3. Comparison of the non-aggregated point cloud from two Velodyne HDL-32e scanners (left) and the aggregated ones (right) on the aggregation period of 1.5s.

We have already estimated the valid transformation for every batch for a short time when the batch's data has been scanned. This transformation corresponds to the IMU position w.r.t. the origin of the local coordinate system. Thus, we have to aggregate one more transformation that expresses the frame difference between the given LiDAR sensor and the IMU reference frame. In this way, we can calculate the final homogeneous transformation transform every single point cloud measurement from the scanner's frame to the local coordinates frame. However, transforming every single point is very demanding on computational power. The points are not transformed immediately, but the batch holds the data in the original frame, and the transformation could be evaluated later in the pipeline.

C. Camera-LiDAR Object Detection

LiDAR can measure the distance and the geometrical shape of the obstacle with high accuracy. On the other hand, to be able to recognize the specific class of the object based only on the point cloud and geometrical shapes is quite challenging. The very opposite of this approach is object detection on the camera images. These days, neural networks can localize and

classify objects on the RGB images in real-time with several dozens of fps [11]. However, although we have quite a reliable object classification and localization in the 2D plane, it is tough to estimate the detected object's distance.

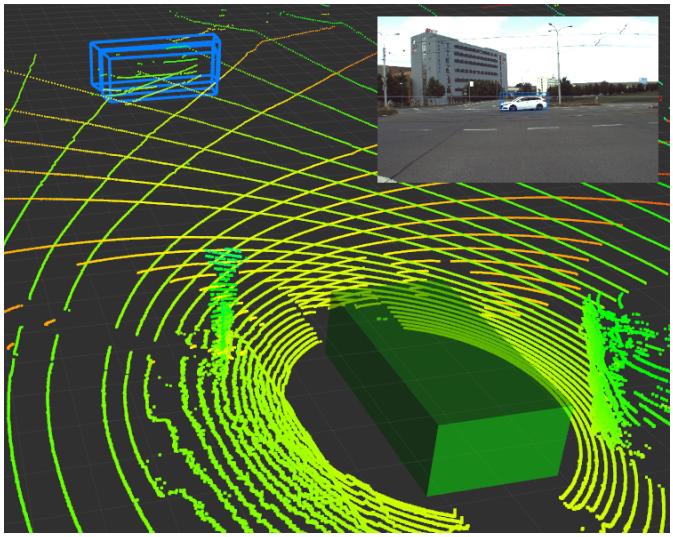

Fig. 4. Car detected by the neural network in both frontal cameras. Distance of the 2D detection is estimated based on the aggregated LiDAR data. Camera view in the right top corner.

For this purpose, we have created a system that fuses the LiDAR data and camera detections and combines them into a single representation.

There is an estimated median distance of the LiDAR measurements projected to the detection bounding box for every detection on the RGB image. This information system generated the 3D frustum representation in the output map of the detected obstacle.

D. RGB YOLO Detections to IR Image

If we focus on the field of neural network training, we can find a large number of papers [12], [13], [14] that deal with object detection on RGB images. However, not many works focus on thermal images [15]. The thermal domain is very beneficial for autonomous agents because it allows agents to sense their surroundings even in bad lights or weather conditions.

There is not only a smaller number of works interested in the learning neural networks to detect objects on the thermal images [16], [17] compared to the visible light spectrum, but also the there is also a dramatically smaller background in existing datasets. There are very few publicly available sources of annotated thermal images that could be used for training purposes, like KAITS [18] or the FLIR [19].

We have proposed a system that would automatically generate a large amount of annotated IR images based on the transferring object detections from the RGB images to the thermal ones, which will help in the future when we will train neural networks for in the thermal image domain [20].

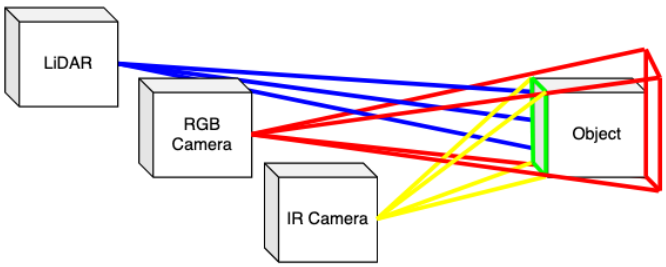

Fig. 5. 1 (red) - the YOLO neural network detects objects in the RGB image. This 2D detection can be represented as a 3D frustum in the real world. 2 (blue) - the LiDAR measures object distance. 3 (green) - by combining LiDAR data and 3D frustum, we can estimate the frontal plane of the detected object. 4 (yellow) - the detected object's plane is reprojected into the IR camera.

The basic idea is to preprocess the detections on the RGB camera, which is semantically close to the IR camera and oriented in the same direction. The nearest IR frame in time is taken for every RGB frame for which the object detection has been performed. In the next phase, the aggregated point cloud model (see IV-E) is used to estimate the distance of the detected obstacle so that the obstacle can be transformed from the 2D image plane into the 3D model of the environment. The last phase is to project the 3D modeled obstacle's frontal face into the thermal image and store the parameters of the projected objects in the same format as the origin RGB detections do.

E. Aggregated LiDAR Data to Image Projection

Currently, many academical publications deal with convolutional neural networks and improve the performance of those state-of-the-art algorithms. However, there is a large number of papers that cover the RGB image object detection, but much less of those that would be dealing with the object classification and detection in the IR (thermal) domain [21] and even less that would try to process the depth images [22].

Our framework allows us to merge all these three domains into a single one.

Every new frame from the thermal camera triggers the following process. From the motion model, the current position of the IMU in the local coordinate system is requested. Simultaneously, the transformation between the IMU and the IR camera is known from the calibration frame.

From the PointCloudAggregator, the currently aggregated set of the point cloud batches is requested and passed into the instance of the DepthMap class. The DepthMap is also provided by the current position and the IMU to camera transformation and the camera calibration parameters. By combining all this information, for every point cloud batch, there is applied additional transformation. The entire transformation chain is currently following from the LiDAR frame to the IMU frame to the Origin frame to the IMU frame to the IR Camera frame.

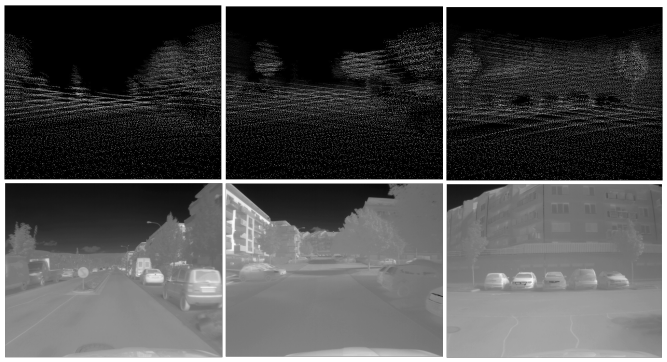

Fig. 6. Example of depth images generated based on the aggregated point cloud model. Depth images (top) paired with the corresponding thermal images (bottom). Point cloud has been projected to the camera frame. The same technique can be applied also on RGB images.

F. Visualizations

The entire mapping process requires a detailed visualization backend to correctly understand every step of the data processing and the final output environment model. For this purpose, we have used RViz - the visualization tool of the ROS toolkit. It supports elementary geometry objects like points or lines and more complex shapes, like arrows, polylines, and complex visualizations, like point clouds, occupancy grids, or the transformation trees. A single class VisualizationHandler processes the visualization that wraps the entire visualization logic.

V. External Dependences

The ROS (Robotic Operating System) [23] makes it possible to communicate with other programs with well-defined API. The entire framework visualization is also realized via the Rviz program, a part of a ROS environment.

For the underlying data representation, like N-dimensional vectors, rotation angles, matrices, quaternions, bounding boxes, frustums, transformations, etc., we have used the previous work of one of the authors, the Robotic Template Library, the C++17 built on the Eigen library. RTL is available at https://github.com/Robotics-BUT/Robotic-Template-Library. Next to the fundamental data primitives representation, RTL also provides several algorithms for point cloud segmentation and vectorization [24], [25]

To solve the assignment problem, we have used the implementation https://github.com/aaron-michaux/munkres-algorithm that refers [26].

VI. Future Work

We have designed our framework in a way that the architecture allows anybody to modify or extend the existing solution. We have put a special effort into building up an abstract system that allows us to scale the current solution to a much larger solution with a reasonable amount of additional complexity. For example, there is no need to modify existing data models and loaders to implement the new sensor's data. We can extend the current software with a few new lines of code based on the given templates. The same we can say about the processing pipelines.

In the future, we are preparing to add more sensors, like radar or ultrasound sensors, extending the current pipeline with the disparity map generation based on the two frontal cameras, optical odometry, or semantic scene segmentation by the neural networks.

We would also like to make this project fully open-source because we believe that these projects can reach a more significant number of developers and researchers, and the bigger community means a more dynamic development process. Our target is to provide a research platform for a large research community that will not need to develop many of those algorithms from scratch and will be able to improve more specific problems for the autonomous robot or the self-driving car domain.

VII. Conclusion

As a result of the research project, we have created the experimental mapping framework that allows easy and fast prototyping of new approaches in autonomous agents. The primary data processing pipeline is a single thread with a waterfall-like architecture, making it easy to understand how the data are processed. The modification does not require complicated code refactoring.

The essential parts of our framework are the precise positioning system that fuses GNSS and IMU data. The LiDAR scans aggregator allows us to integrate multiple point clouds into a single dense environment model. Next is the point cloud to camera projection and depth image generating, the point cloud obstacle detection, YOLO neural network-based 3D obstacle detection, and RGB to IR neural network detection mapping.

References

[1] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[2] W. Li, C. Pan, R. Zhang, J. Ren, Y. Ma, J. Fang, F. Yan, Q. Geng, X. Huang, H. Gong *et al.*, "Aads: Augmented autonomous driving simulation using data-driven algorithms," *arXiv preprint arXiv:1901.07849*, 2019.

[3] A. Ligocki, A. Jelinek, and L. Zalud, "Brno urban dataset–the new data for self-driving agents and mapping tasks," *arXiv preprint arXiv:1909.06897*, 2019.

[4] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 year, 1000 km: The oxford robotcar dataset," *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 3–15, 2017.

[5] F. Caron, E. Duflos, D. Pomorski, and P. Vanheeghe, "Gps/imu data fusion using multisensor kalman filtering: introduction of contextual aspects," *Information fusion*, vol. 7, no. 2, pp. 221–230, 2006.

[6] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[7] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[8] G. A. Terejanu, "Discrete kalman filter tutorial," *University at Buffalo, Department of Computer Science and Engineering, NY*, vol. 14260, 2013.

[9] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, "Lidar point clouds correction acquired from a moving car based on can-bus data," *arXiv preprint arXiv:1706.05886*, 2017.

[10] B. Zhang, X. Zhang, B. Wei, and C. Qi, "A point cloud distortion removing and mapping algorithm based on lidar and imu ukf fusion," in *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2019, pp. 966–971.

[11] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[12] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[14] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[15] K. Agrawal and A. Subramanian, "Enhancing object detection in adverse conditions using thermal imaging," *arXiv preprint arXiv:1909.13551*, 2019.

[16] M. Ivašić-Kos, M. Krišto, and M. Pobar, "Human detection in thermal imaging using yolo," in *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*, 2019, pp. 20–24.

[17] C. Herrmann, M. Ruf, and J. Beyerer, "Cnn-based thermal infrared person detection by domain adaptation," in *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, vol. 10643. International Society for Optics and Photonics, 2018, p. 1064308.

[18] J. Jeong, Y. Cho, Y.-S. Shin, H. Roh, and A. Kim, "Complex urban dataset with multi-level sensors from highly diverse urban environments," *The International Journal of Robotics Research*, p. 0278364919843996, 2019.

[19] Tech. Rep., also available as https://www.flir.in/oem/adas/adas-dataset-form/.

[20] L. Y. Pratt, "Discriminability-based transfer between neural networks," in *Advances in neural information processing systems*, 1993, pp. 204–211.

[21] C. D. Rodin, L. N. de Lima, F. A. de Alcantara Andrade, D. B. Haddad, T. A. Johansen, and R. Storvold, "Object classification in thermal images using convolutional neural networks for search and rescue missions with unmanned aerial systems," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[22] T. Ophoff, K. Van Beeck, and T. Goedemé, "Exploring rgb+ depth fusion for real-time object detection," *Sensors*, vol. 19, no. 4, p. 866, 2019.

[23] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[24] A. Jelinek, L. Zalud, and T. Jilek, "Fast total least squares vectorization," *J. Real-Time Image Process.*, vol. 16, no. 2, p. 459–475, Apr. 2019. [Online]. Available: https://doi.org/10.1007/s11554-016-0562-6

[25] A. Jelinek and L. Zalud, "Augmented postprocessing of the ftls vectorization algorithm," in *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics*, ser. ICINCO 2016. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2016, p. 216–223. [Online]. Available: https://doi.org/10.5220/0005962902160223

[26] R. Pilgrim, "Tutorial on implementation of munkres' assignment algorithm," 08 1995, p. 13.